CS671 – Deep Learning and Applications
Feb-June 2021
Programming Assignment 1
Report
Group 09

Ashutosh (B18010) Jahnvi (B18060) Anuj (B18161)

CLASSIFICATION

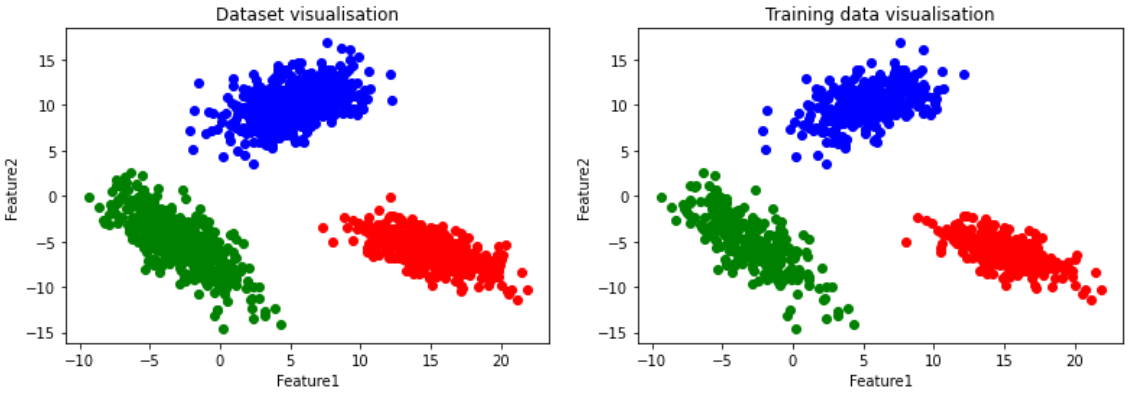
Dataset 1 (a): 2-dimensional artificial data of 3 classes with 500 points in each class.
Pre Training: Since the range of data is fairly low, we haven't normalized the data.+

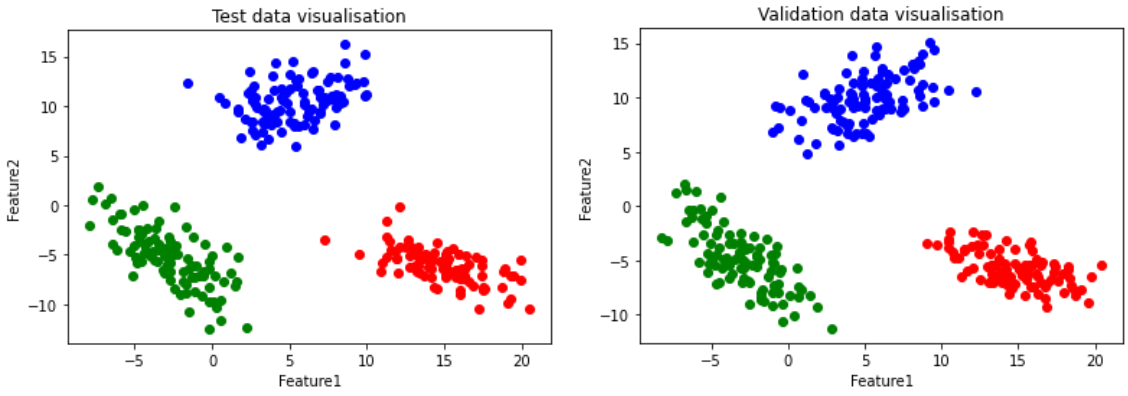
Training Model: Perceptron

First, we divided data randomly into training, test, and validation data with following proportion:

- Training: 60%
- Validation: 20%
- Test: 20%

Data Visualisation





Since it is a **multiclass classification**, **One-against-one** Approach is used for training.
Number of classes (N) = 3
Number of perceptrons = N(N-1)/2 = 3

Training:
Mode of Training: Pattern Mode

Weight initialization: Weights are generated randomly from the standard normal distribution.

Parameter Learning: Parameter (w) learning is done by minimizing the error using the Gradient descent method.
To ensure that the learning rate is large in beginning and smaller later, we have used the following values for the learning rate:

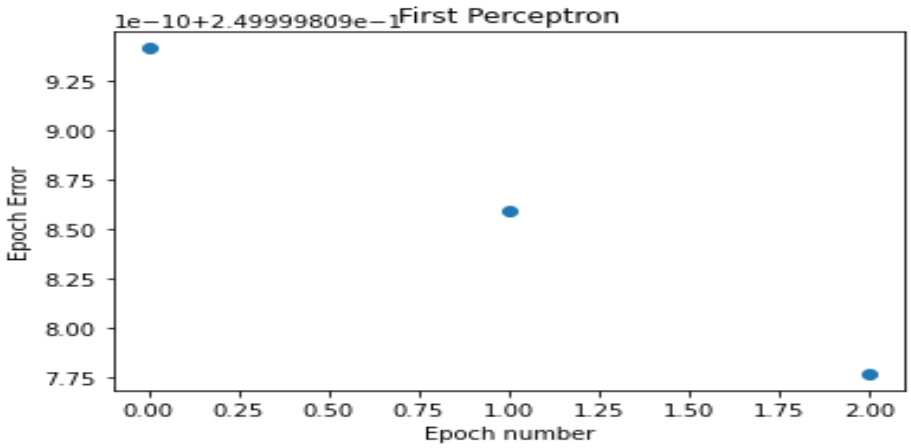
epoch_number<50	epoch_number<200	else
learning rate = 0.1	learning rate = 0.001	learning rate = 0.0001

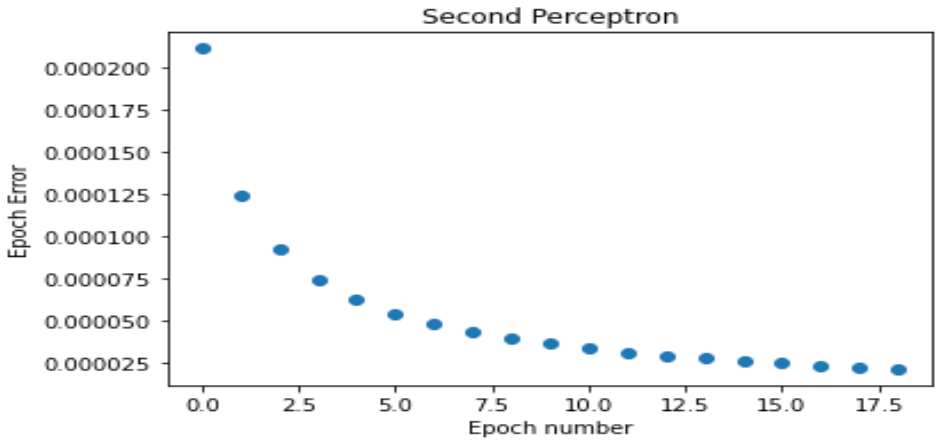
Activation function: Logistic Sigmoidal Function
Logistic function: $f(a) = 1/(1+e^{-\beta a})$, where β is a slope parameter
We have used $\beta = 1$.

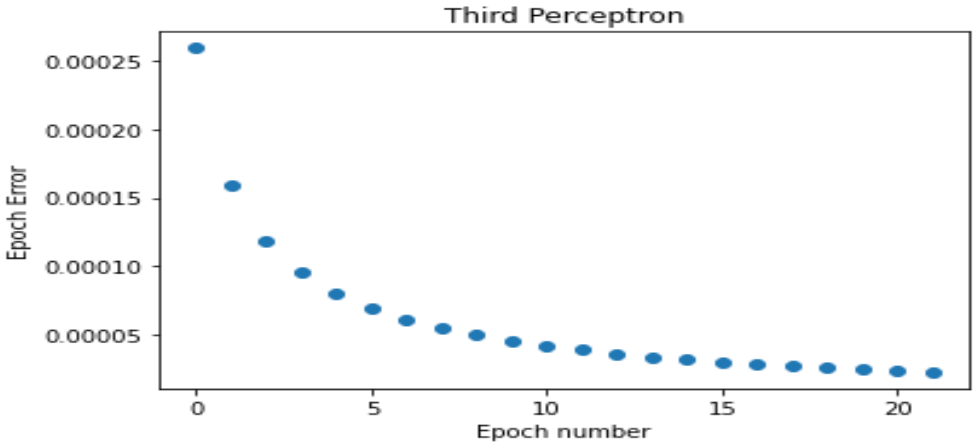
Stopping Criteria: Threshold on the difference between average error in consecutive epochs, which taken as 0.000001.

For 3 perceptrons training, the average epoch error with epoch number is plotted as follows:

Average Epoch Error vs Epoch Number

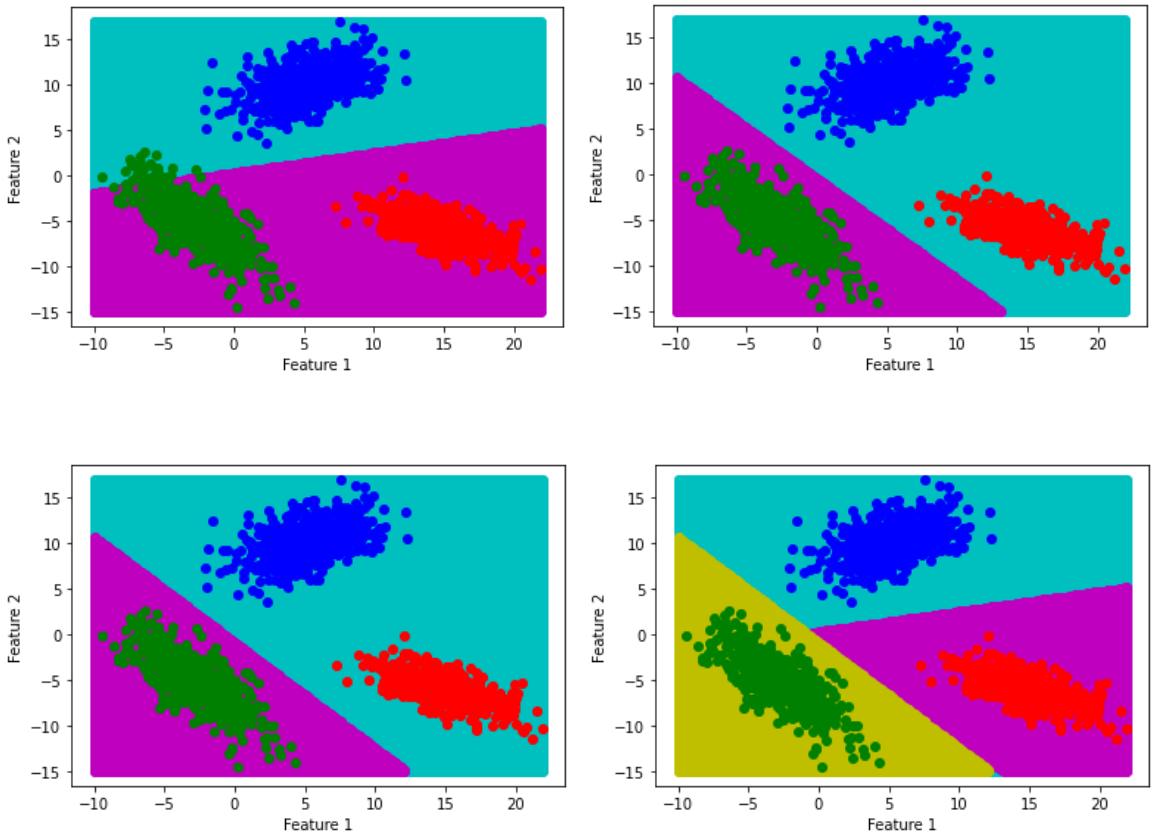






After Training, we plotted **decision plots** to see the decision region for each class learned by each perceptron and combined them for all three classes:

Decision Plots



Finally, we evaluated our model with **confusion matrix** and **accuracy** for each of training, validation, and test data which came out as follows:

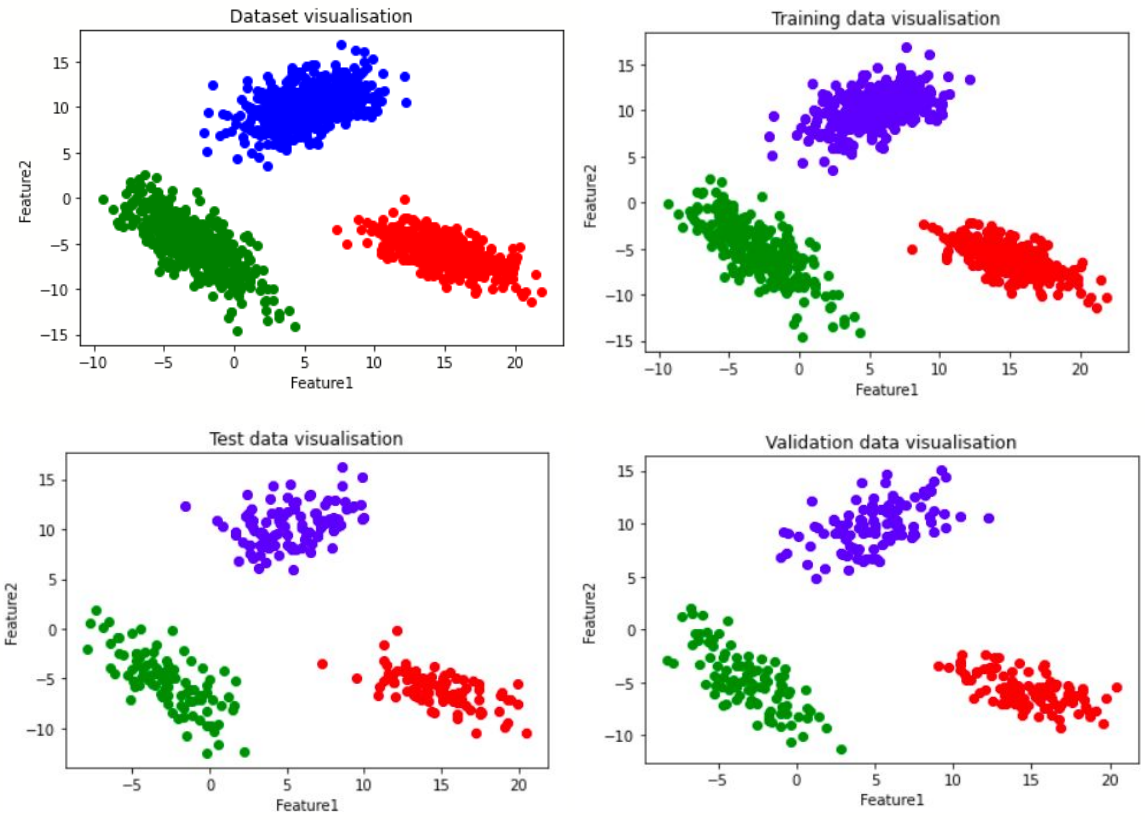
	Training Data	Validation Data	Test Data
Confusion Matrix	[[300 0 0] [0 300 0] [0 0 300]]	[[300 0 0] [0 300 0] [0 0 300]]	[[300 0 0] [0 300 0] [0 0 300]]
Accuracy	100%	100%	100%

Training Model: MLFFNN (1 hidden layer)

First, we divided data randomly into training, test, and validation data with following proportion:

- Training: 60%
- Validation: 20%
- Test: 20%

Data Visualisation



Training:

Mode of Training: Pattern Mode

Weight initialization: Weights are generated randomly from the standard normal distribution.

Parameter Learning: Parameters (w_{ij} , w_{jk}) learning is done by minimizing the error using the Gradient descent method.
To ensure that the learning rate is large in beginning and smaller later, we have used the following values for the learning rate:

epoch_number<50	epoch_number<200	else
learning rate = 0.1	learning rate = 0.001	learning rate = 0.0001

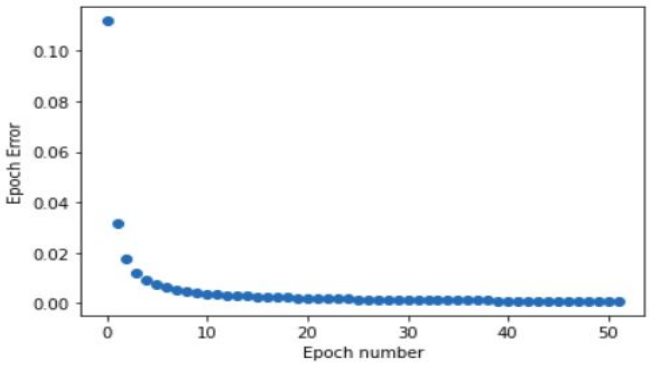
Activation function: Logistic Sigmoidal Function: $f(a) = 1/(1+e^{-\beta a})$,
where β is a slope parameter. We have used $\beta = 1$.

Stopping Criteria: Threshold on the difference between average error in consecutive epochs, which taken as 0.000001.

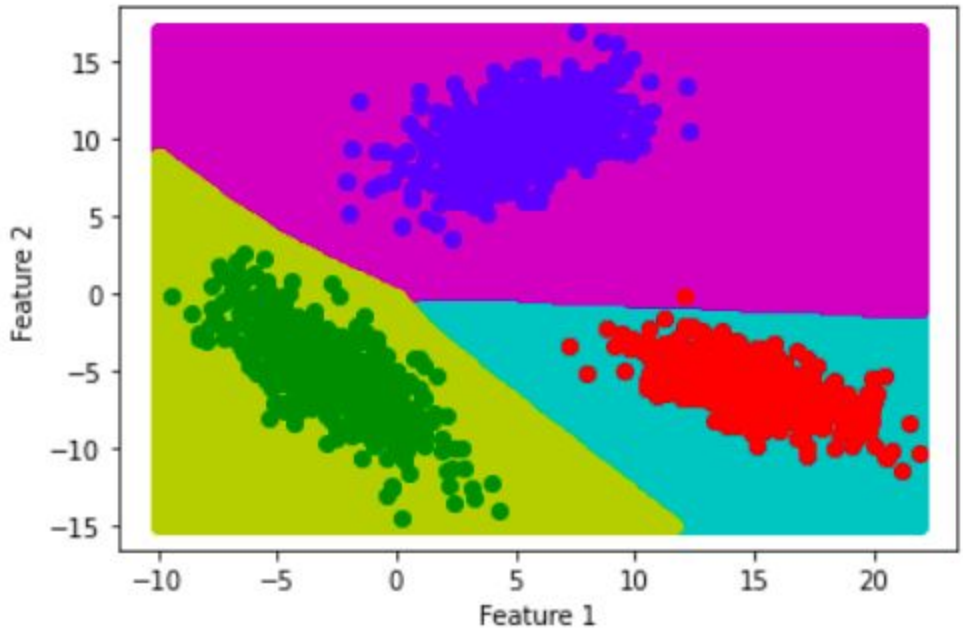
Cross-Validation:
Taking number of nodes in hidden layer(J) = 3
Confusion Matrix for validation data : [[100. 0. 0.]
[0. 100. 0.]
[0. 0. 100.]]

Accuracy for validation data is: 100.0%
Considering, 100% accuracy with validation data, J = 3 is considered.

Average Epoch Error vs Epoch Number



Decision Plot

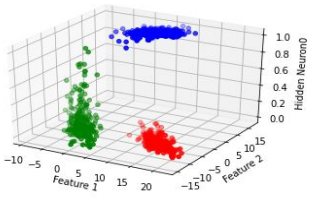


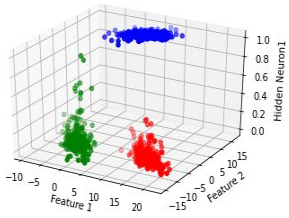
Evaluation for training and test data:

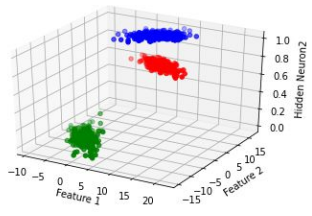
	Training Data	**Test Data**
Confusion Matrix	[[300. 0. 0.] [0. 300. 0.] [0. 0. 300.]]	[[99. 1. 0.] [0. 100. 0.] [0. 0. 100.]]
Accuracy	100%	99.6667%

Output of all the neurons:

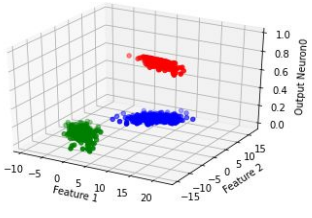
- Hidden Neurons:

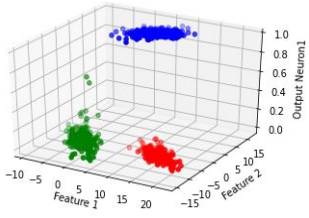


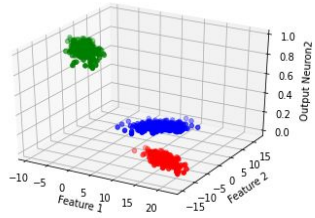




- Output Neurons:







Performance Comparison:

Accuracy	**Training Data**	**Test Data**	**Validation Data**
Perceptron	100	100	100
MLFFNN	100	99.667	100

Perceptron gives better accuracy than MLFFNN and also it takes fewer computations as compared to MLFFNN. Hence, the **perceptron** is better for Linearly Separable classes.

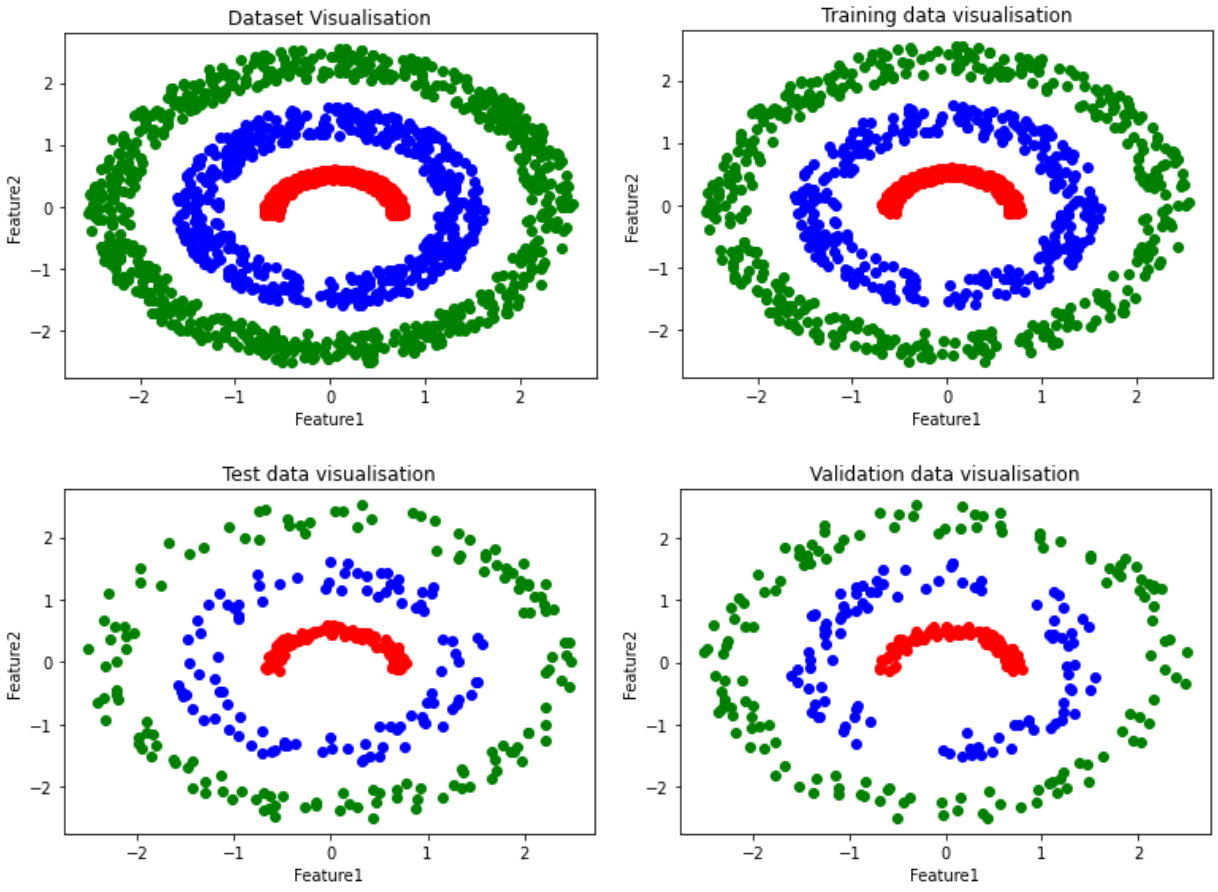
Dataset 1 (b): 2-dimensional artificial data of 3 classes with 500 points in the first 2 classes and 700 points in the 3rd class.

Pre Training: Since the range of data is fairly low, we haven't normalized the data.

Training Model: Perceptron

First, we divided data randomly into training, test, and validation data with following proportion:

- Training: 60%
- Validation: 20%
- Test: 20%



Since it is a **multiclass classification**, **One-against-one** Approach is used for training.

Number of classes (N) = 3

Number of perceptrons = N(N-1)/2 = 3

Training:
Mode of Training: Pattern Mode

Weight initialization: Weights are generated randomly from the standard normal distribution.

Parameter Learning: Parameter (w) learning is done by minimizing the error using the Gradient descent method.

To ensure that the learning rate is large in beginning and smaller later, we have used the following values for the learning rate:

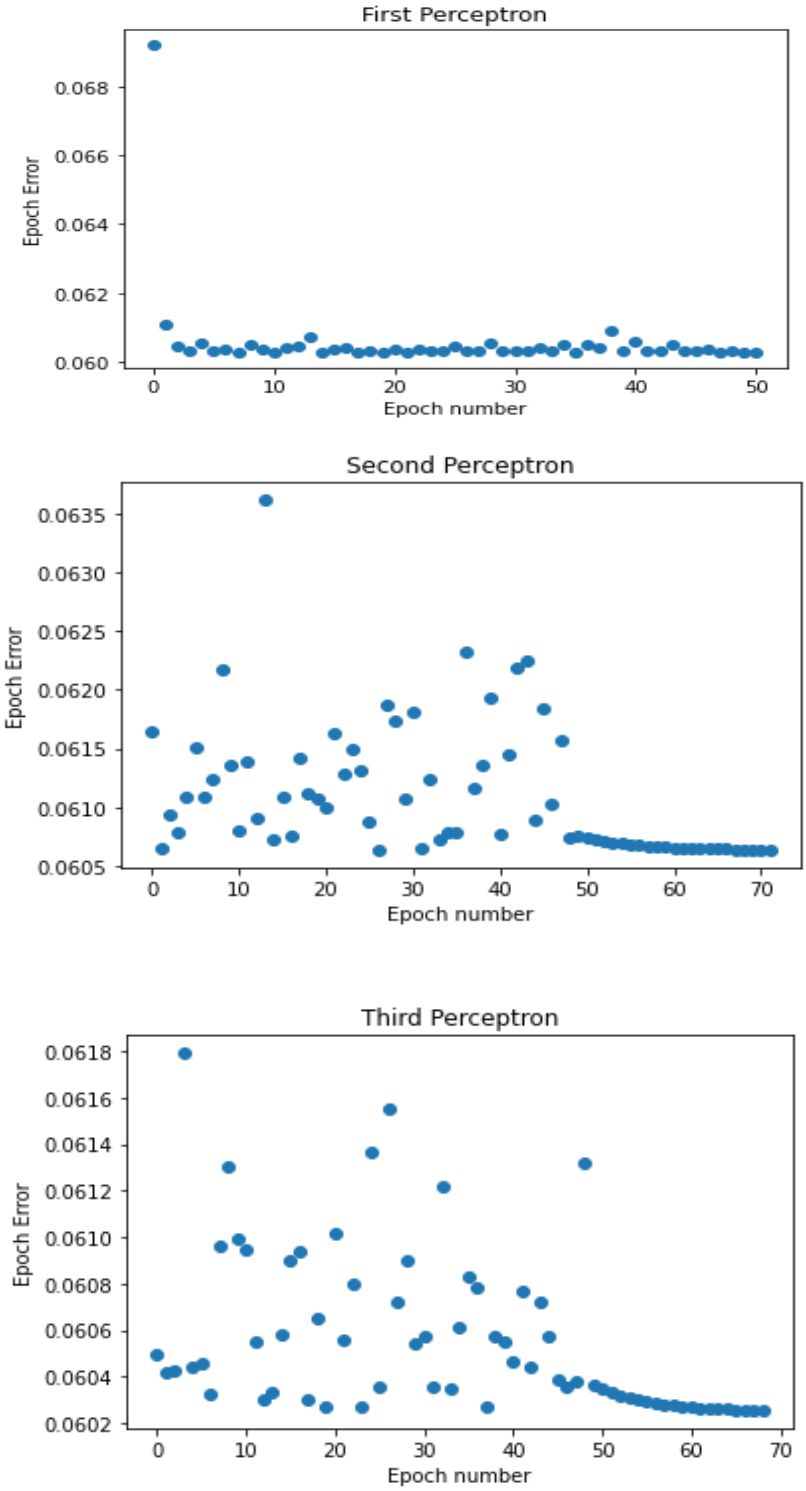
epoch_number<50	epoch_number<200	else
learning rate = 0.1	learning rate = 0.001	learning rate = 0.0001

Activation function: Logistic Sigmoidal Function
Logistic function: $f(a) = 1/(1+e^{-\beta a})$, where β is a slope parameter
We have used $\beta = 1$.

Stopping Criteria: Threshold on the difference between average error in consecutive epochs, which is taken as 0.000001.

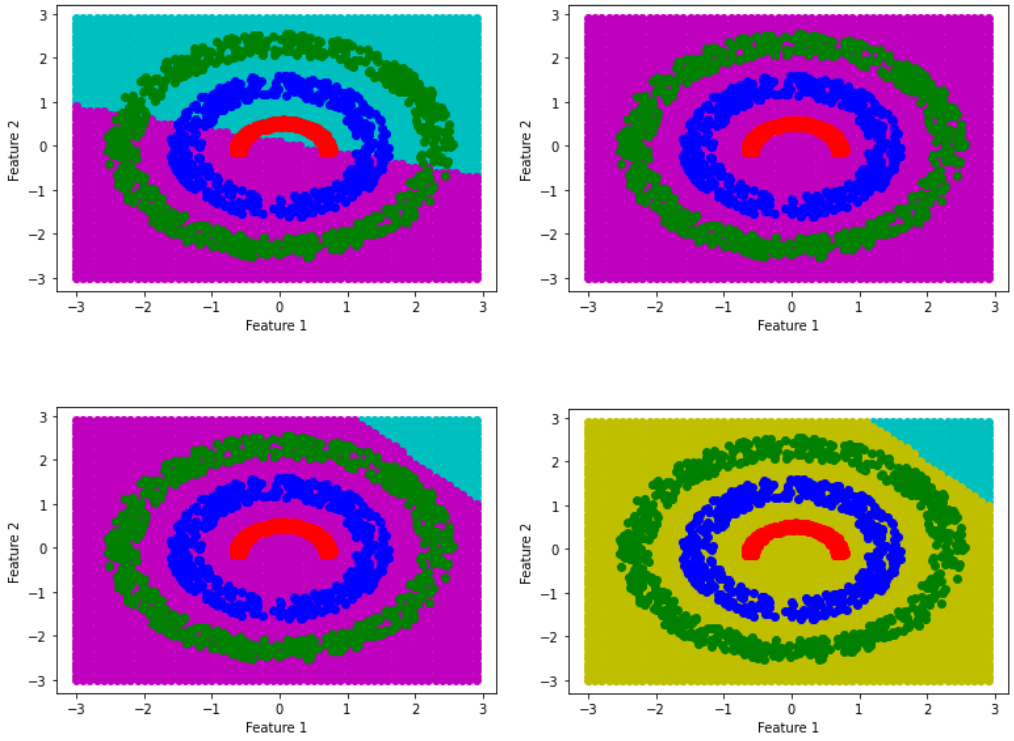
For 3 perceptrons training, the average epoch error with epoch number is plotted as follows:

Average Epoch Error vs Epoch Number



After Training, we plotted **decision plots** to see the decision region for each class learned by each perceptron and combined them for all three classes:

Decision Plots



Finally, we evaluated our model with **confusion matrix** and **accuracy** for each of training, validation, and test data which came out as follows:

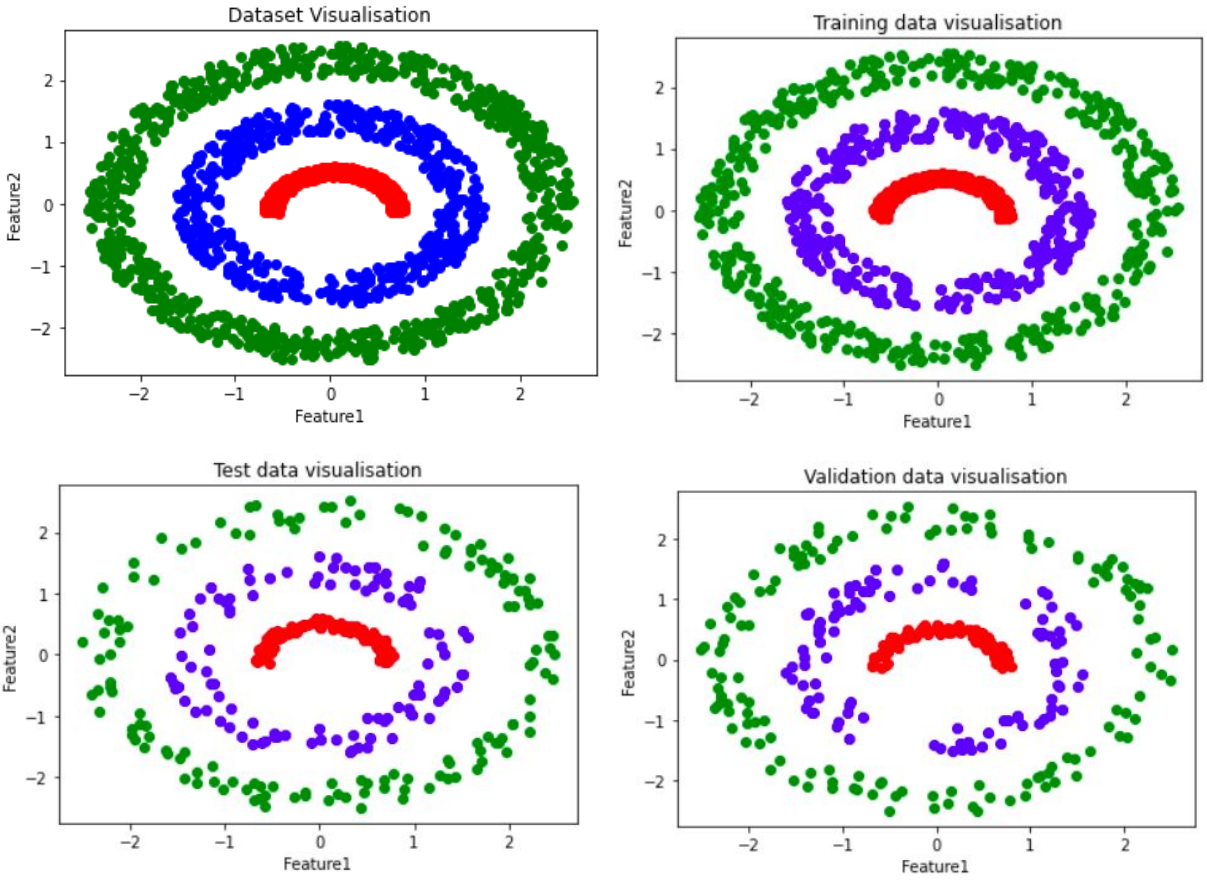
	Training Data	Validation Data	Test Data
Confusion Matrix	[[0 0 300] [0 0 300] [0 0 420]]	[[0 0 100] [0 0 100] [0 0 140]]	[[0 0 100] [0 0 100] [0 0 140]]
Accuracy	41.176%	41.176%	41.176%

Training Model: MLFFNN (1 hidden layer)

First, we divided data randomly into training, test and validation data with following proportion:

- Training: 60%
- Validation: 20%
- Test: 20%

Data Visualisation



Training:
Mode of Training: Pattern Mode

Weight initialization: Weights are generated randomly from the standard normal distribution.

Parameter Learning: Parameters (w_{ij} , w_{jk}) learning is done by minimizing the error using the Gradient descent method.
To ensure that the learning rate is large in beginning and smaller later, we have used the following values for the learning rate:

epoch_number<50	epoch_number<200	else
learning rate = 0.1	learning rate = 0.001	learning rate = 0.0001

Activation function: Logistic Sigmoidal Function
Logistic function: $f(a) = 1/(1+e^{-\beta a})$, where β is a slope parameter
We have used $\beta = 1$.

Stopping Criteria: Threshold on the difference between average error in consecutive epochs, which taken as 0.000001.

Cross-Validation:

- Taking number of nodes in hidden layer(J) = 3
  Confusion Matrix for validation data : [[100. 0. 0.]
  [34. 2. 64.]
  [0. 0. 140.]]
  Accuracy for validation data is: 71.1764%

- Taking J = 4
  Confusion Matrix for validation data : [[100. 0. 0.]
  [31. 23. 46.]
  [0. 0. 140.]]
  Accuracy for validation data is : 77.3529%

- Taking J = 6
  Confusion Matrix for validation data : [[100. 0. 0.]
  [9. 68. 23.]
  [0. 0. 140.]]
  Accuracy for validation data is : 90.58%

- Taking J = 8
  Confusion Matrix for validation data : [[100. 0. 0.]
  [18. 59. 23.]
  [0. 0. 140.]]

Accuracy for validation data is : 87.941%

- Taking J = 12

  Confusion Matrix for validation data : [[100. 0. 0.]
  [14. 63. 23.]
  [0. 0. 140.]]

  Accuracy for validation data is : 89.117%

- Taking J = 16

  Confusion Matrix for validation data : [[100. 0. 0.]
  [11. 76. 13.]
  [0. 0. 140.]]

  Accuracy for validation data is : 92.9412%

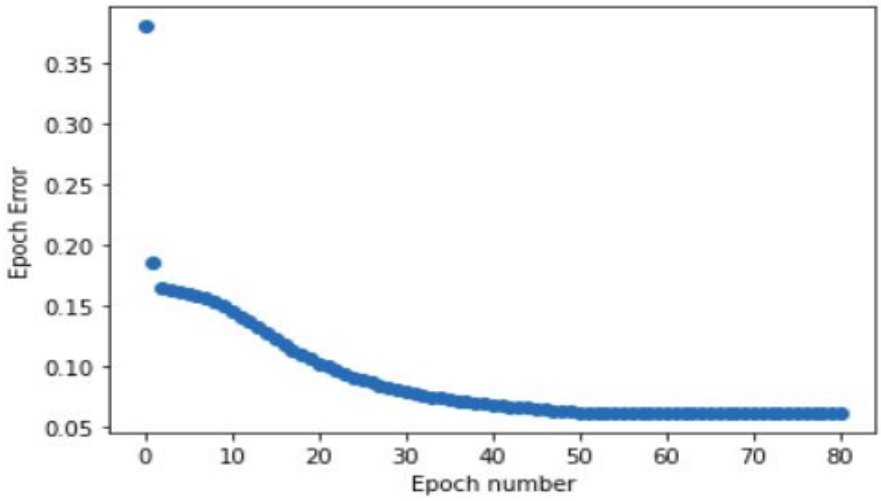
- Taking J = 30

  Confusion Matrix for validation data : [[100. 0. 0.]
  [12. 71. 17.]
  [0. 0. 140.]]

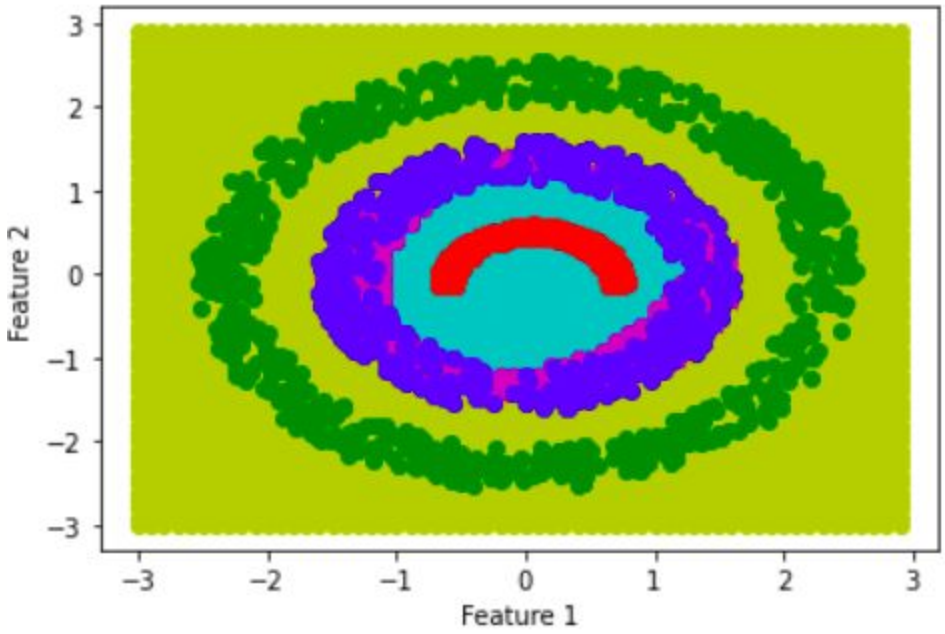
  Accuracy for validation data is : 91.470%

Considering, no significant improvement in accuracy after 16 neurons, **J = 16** is considered.

Average Epoch Error vs Epoch Number



Decision Plot

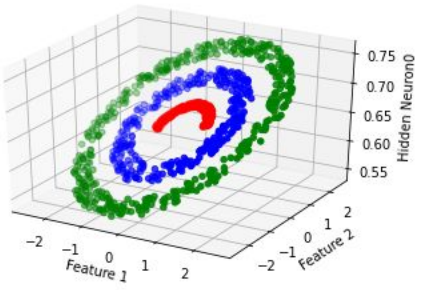


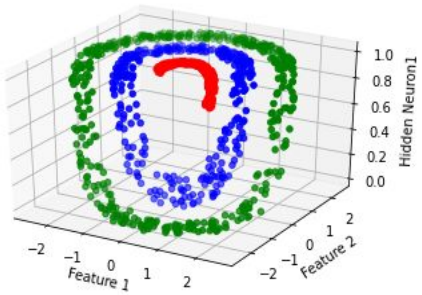
Evaluation for training and test data:

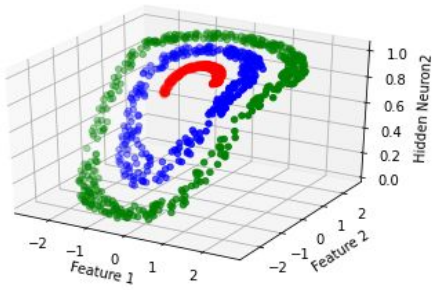
	Training Data	Test Data
Confusion Matrix	[[300. 0. 0.] [36. 225. 39.] [0. 0. 420.]]	[[100. 0. 0.] [12. 71. 17.] [0. 0. 140.]]
Accuracy	92.647%	91.4705%

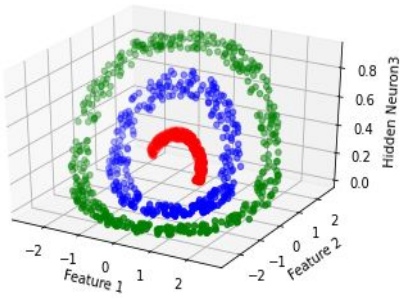
Output of each neuron after training:

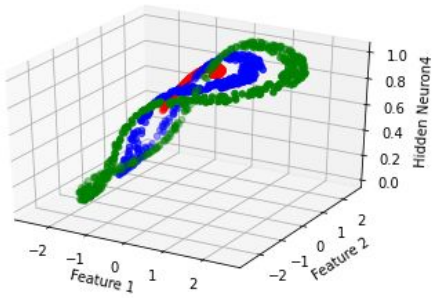
- Hidden layer neurons:

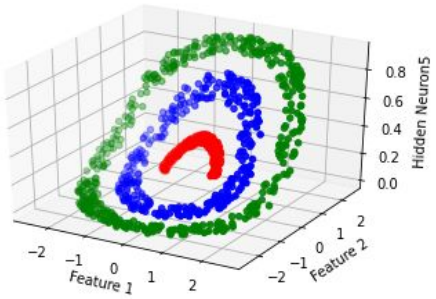


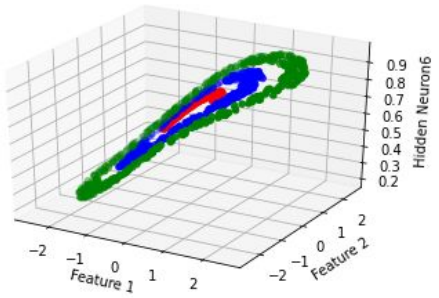


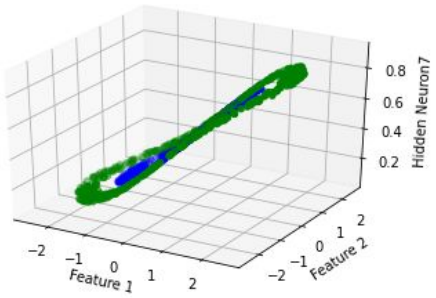


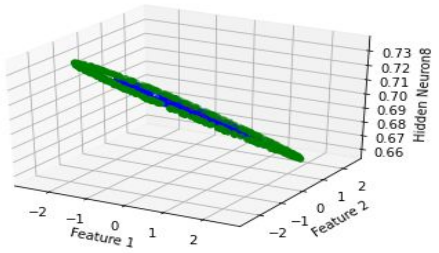


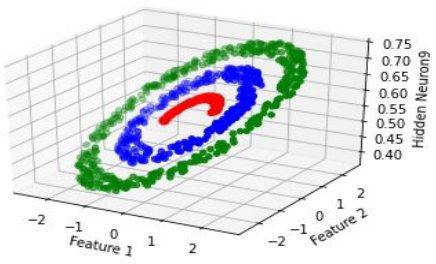

Hidden Neuron4
Feature 1
Feature 2

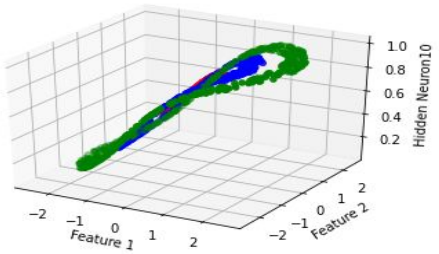

Hidden Neuron5
Feature 1
Feature 2

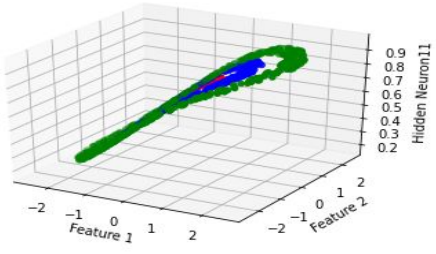

Hidden Neuron6
Feature 1
Feature 2

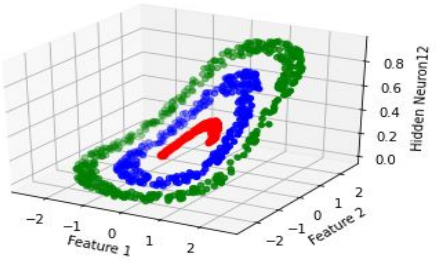

Hidden Neuron7
Feature 1
Feature 2

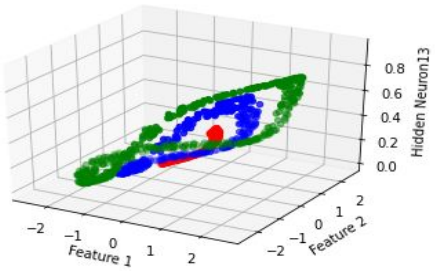

Hidden Neuron8
Feature 1
Feature 2

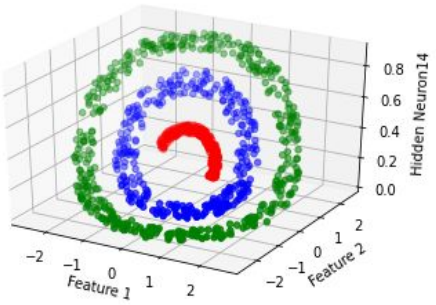

Hidden Neuron9
Feature 1
Feature 2

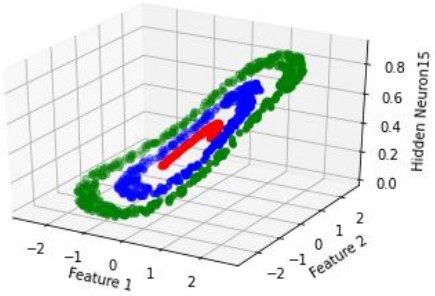

Hidden Neuron10
Feature 1
Feature 2


Hidden Neuron11
Feature 1
Feature 2

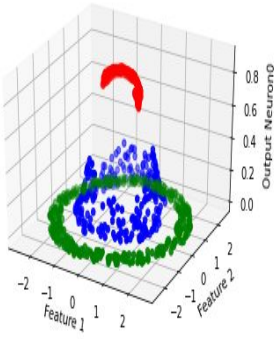


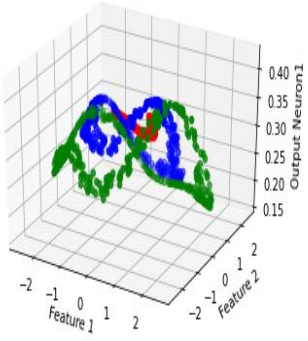


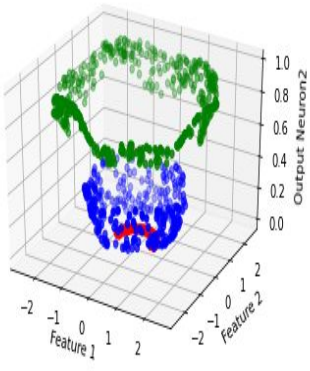




- **Output Neuron**







Training Model: MLFFNN (2 hidden layers)
Training:
Mode of Training: Pattern Mode

Weight initialization: Weights are generated randomly from the standard normal distribution.

Parameter Learning: Parameters (w_{ij1} , w_{j1j2}, w_{j2k}) learning is done by minimizing the error using the Gradient descent method.
Since, it takes large number of epochs to reach minima, learning rate is set as 0.1:

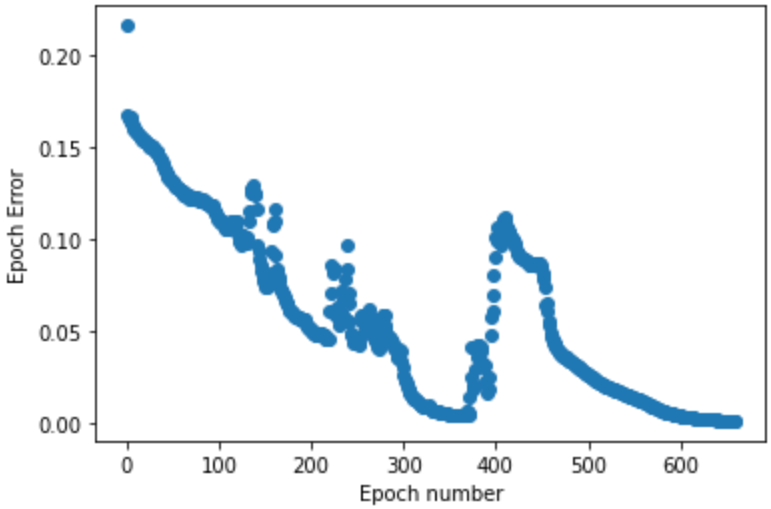
Activation function: Logistic Sigmoidal Function
Logistic function: $f(a) = 1/(1+e^{-\beta a})$, where β is a slope parameter
We have used $\beta = 1$.

Stopping Criteria: Threshold on the average error, which taken as 0.001.

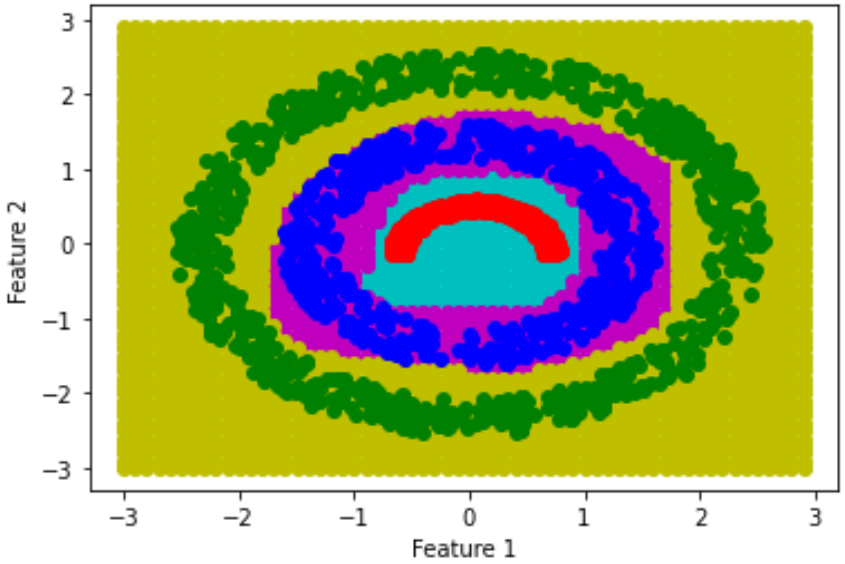
Cross-Validation:
Taking number of nodes in 1st hidden layer(J1) = 6
Taking number of nodes in 2nd hidden layer(J2) = 6

Confusion Matrix for validation data : [[100. 0. 0.]
[0. 100. 0.]
[0. 0. 140.]]
Accuracy for validation data is: 100.0%
Considering, 100% accuracy with validation data, J1 = 6, J2 = 6 is considered.

Average Epoch Error vs Epoch Number



Decision Plot

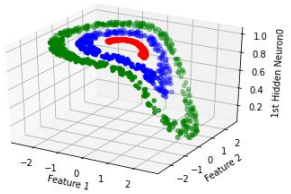


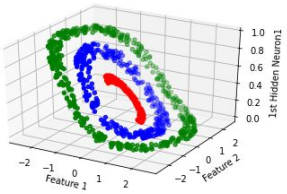
Evaluation for training and test data:

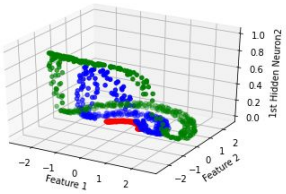
	Training Data	Test Data
Confusion Matrix	[[300. 0. 0.] [0. 300. 0.] [0. 0. 420.]]	[[100. 0. 0.] [0. 100. 0.] [0. 0. 140.]]
Accuracy	100%	100%

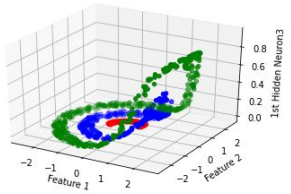
Output of all the neurons:

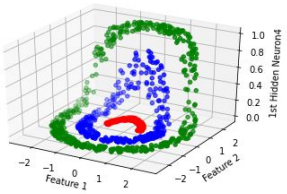
- 1st Hidden Layer

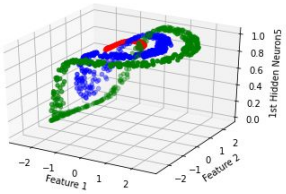




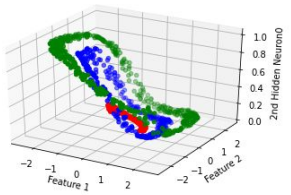


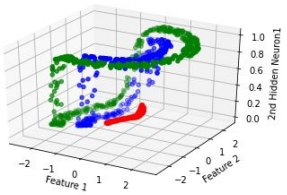


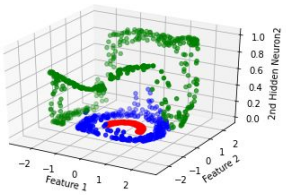


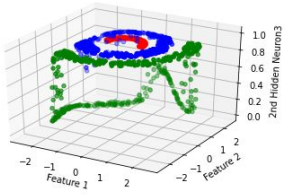


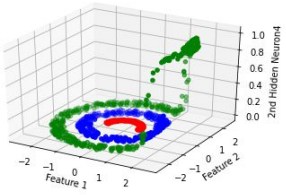
- 2nd Hidden Layer

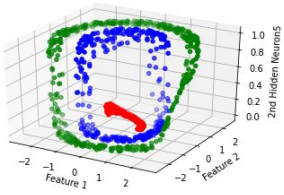




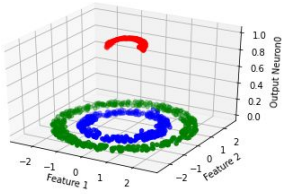


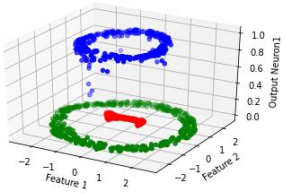


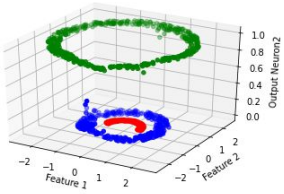




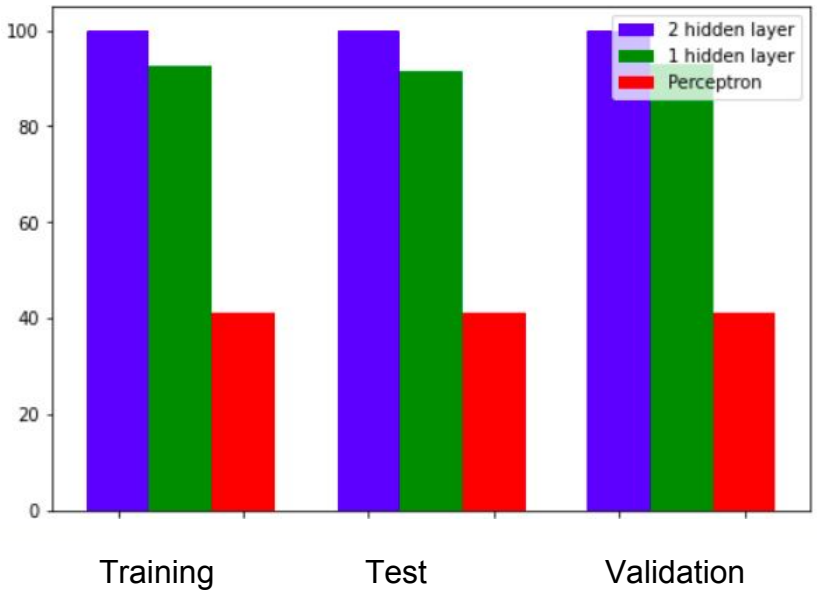
- Output Layer







Model Comparison:



In case non linearly separable classification, 2 hidden layers perform the best giving 100% accuracy and perceptron performs the worst classifying every data point in a single class.

D**ataset 2:** Scene image data corresponding to 3 different classes. Train set consists of 50 images of each class which is divided into 40 images for training and 10 images for validation, the test set consists of 50 images of each class.

Pre Training:

For pre training we **extracted bag of visual words(bovw)** feature for each image using the following algorithm:

1. Row and column of each image from test and training was increased to the nearest multiple of 32 by copying pixels from the start of the image.
2. Then we divided each image into **32*32 pixels patches** from test and training data.
3. Then we extracted **8 bin color histograms** for each patch and for each color. Thus, we got 3, 8 dimensional vectors for each patch.
4. We concatenated those **3, 8 dimensional vectors into 24 dimensional vectors** for each patch of each image from test and training data.
5. Then we applied **k-means for 32 clusters** using the patches 24 dimensional vectors.
6. Then we started to again divide images in patches and for each image counted the number of patches of that image which lie in each cluster.
7. **Then we got a 32 dimensional vector where each entry represents the number of patches from that image lying in that particular cluster.**
8. Then to normalise it we divided this vector with the total number of patches.
9. **Thus we got BoVW representation of each image from test and training data into a 32 dimensional vector.**
10. At the end we added one more column to each image which represented the class of that image such as:
    Class 0: auditorium
    Class 1: desert_vegetation
    Class 2: synagogue_outdoor

Training Model: MLFFNN (1 hidden layer)
Training:
Mode of Training: Pattern Mode

Weight initialization: Weights are generated randomly from the standard normal distribution.

Parameter Learning: Parameters (w_{ij} , w_{jk}) learning is done by minimizing the error using the Gradient descent method.
Since, it will take large number of iterations to converge, learning_rate is chosen as 0.1.

Activation function: Logistic Sigmoidal Function
Logistic function: $f(a) = 1/(1+e^{-\beta a})$, where β is a slope parameter
We have used $\beta = 1$.

Stopping Criteria: Threshold on the average error, which taken as 0.05

Cross-Validation:
Taking number of nodes in hidden layer(J) = 32
Confusion Matrix for validation data : `[[8. 1. 1.]`
`[0. 7. 3.]`
`[1. 1. 8.]]`
Accuracy for validation data is: `76.666`%

Confusion Matrix for testing data : [[31. 12. 7.]
[8. 30. 12.]
[10. 5. 35.]]
Accuracy for test data is : 64.0

Confusion Matrix for training data : [[38. 1. 1.]
[3. 35. 2.]
[4. 2. 34.]]
Accuracy for training data is : 89.16666666666667

Here, we see that due to small data, the **data overfits** and learning is not good. If we increase the model complexity, overfitting increases and learning is not good.

REGRESSION

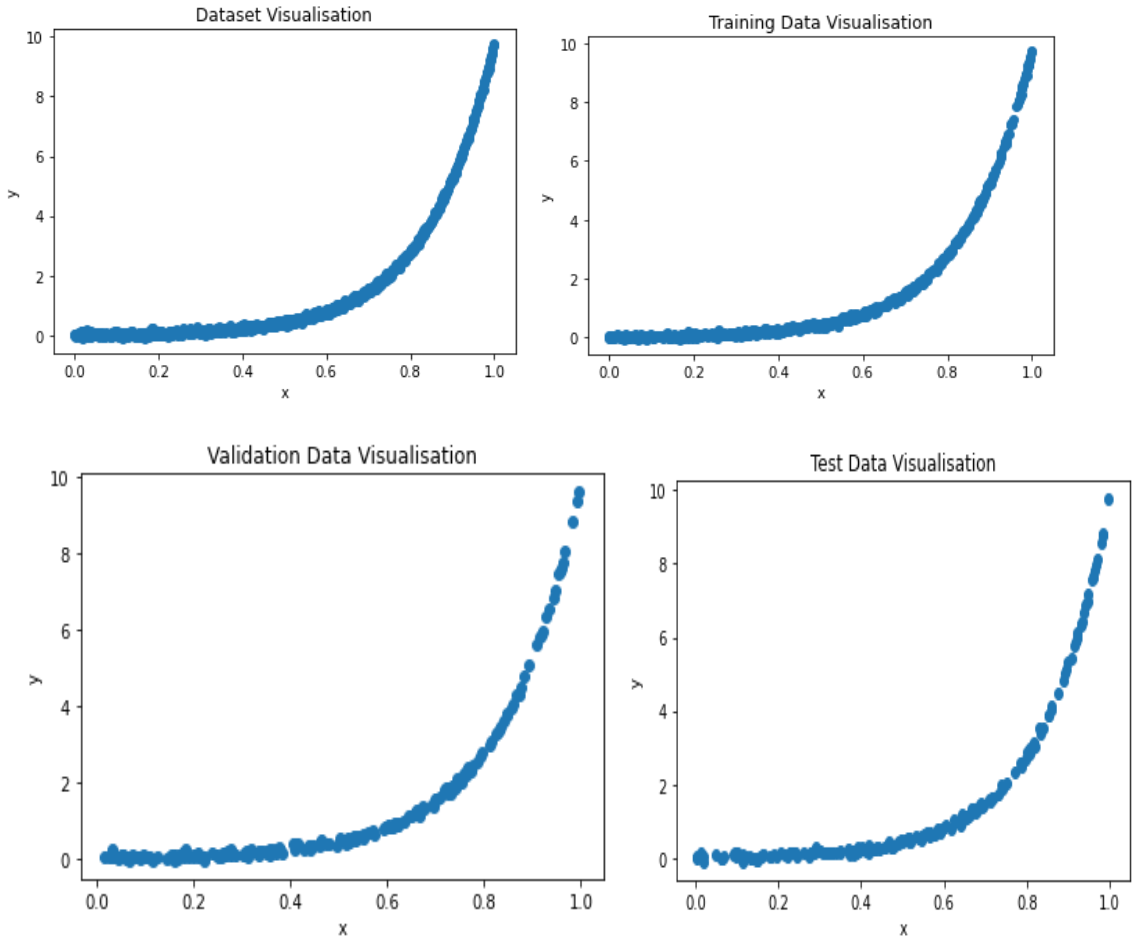
Dataset 1: 1-dimensional (Univariate) input and 1-dimensional output

Pre Training*:* Since the range of data is fairly low, we haven't normalized the data.

Training Model: Perceptron

First, we divided data randomly into training, test, and validation data with following proportion:

- Training: 60%
- Validation: 20%
- Test: 20%



Activation function: Linear Function

Linear function: $f(a) = a$

Training:

Mode of Training: Pattern Mode

Weight initialization: Weights are generated randomly from the standard normal distribution.

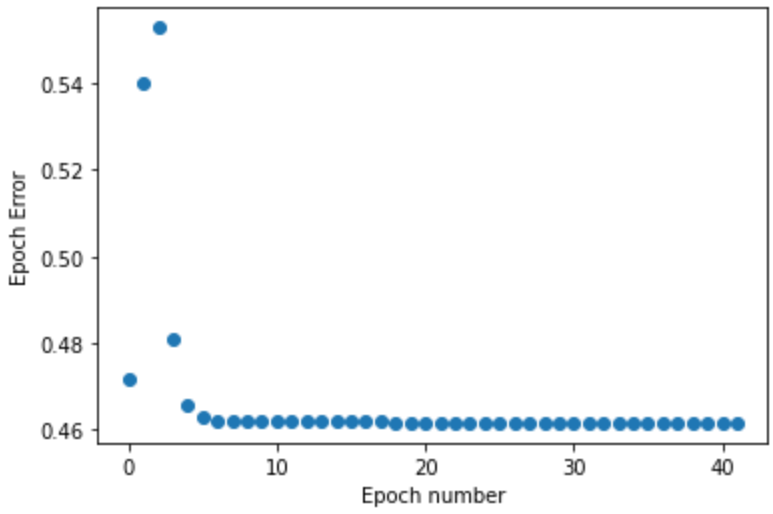
Parameter Learning: Parameter (w) learning is done by minimizing the error using the Gradient descent method.

To ensure that the learning rate is large in beginning and smaller later, we have used the following values for the learning rate:

epoch_number<50	epoch_number<200	else
learning rate = 0.1	earning rate = 0.001	learning rate = 0.0001

Activation function: Linear Function
Stopping Criteria: Threshold on the average error, which taken as 0.000001.
For 3 perceptrons training, the average epoch error with epoch number is plotted as follows:

Average Epoch Error with Epoch Number for perceptron training

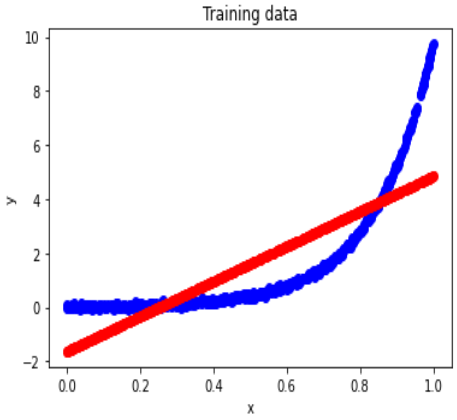


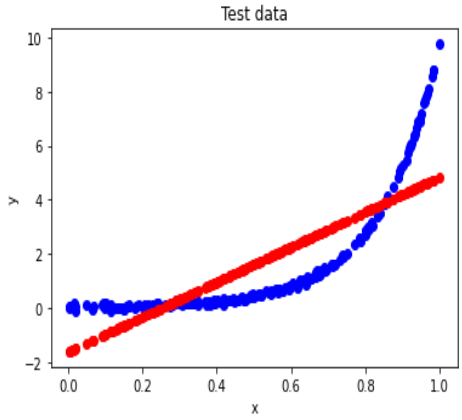
Mean Squared Error (MSE) on training data, validation data and test data:

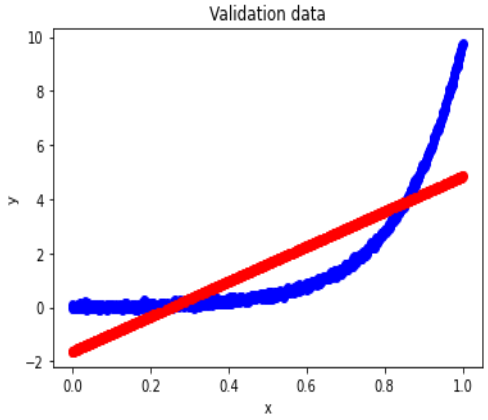
Training data	Validation data	Test data
0.4616044983269184	0.40582269781927927	0.48095718468635557

Plots of the prediction made by model for training, test and validation data:

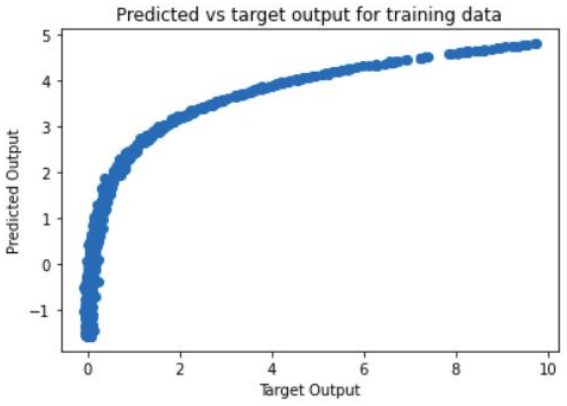
- Red colour indicates model prediction
- Blue colour indicates target output

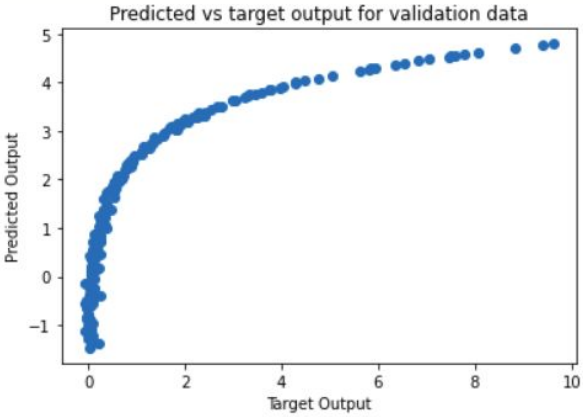


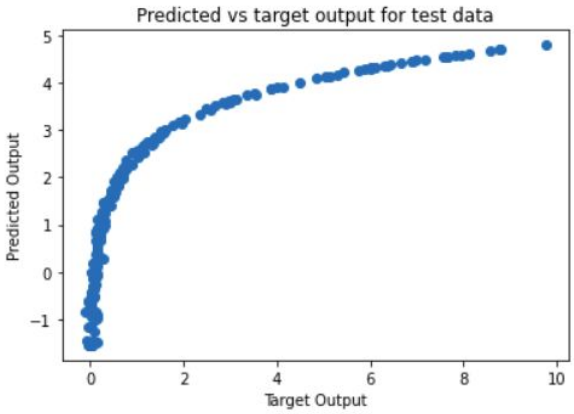




Plots of Predicted Output with Target Output:



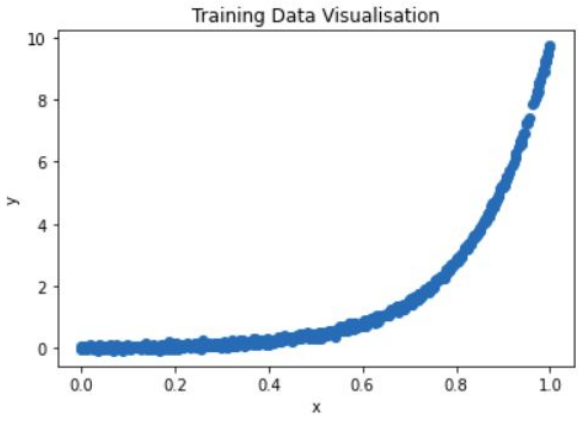


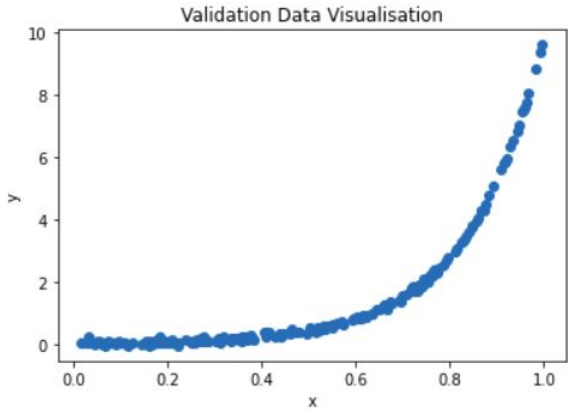


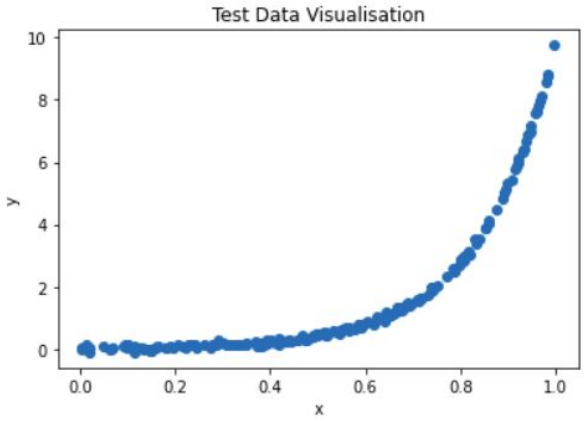
Training Model: MLFFNN (1 hidden layer)

First, we divided data randomly into training, test and validation data with following proportion:

- Training: 60%
- Validation: 20%
- Test: 20%







Training:

Mode of Training: Pattern Mode

Weight initialization: Weights are generated randomly from the standard normal distribution.

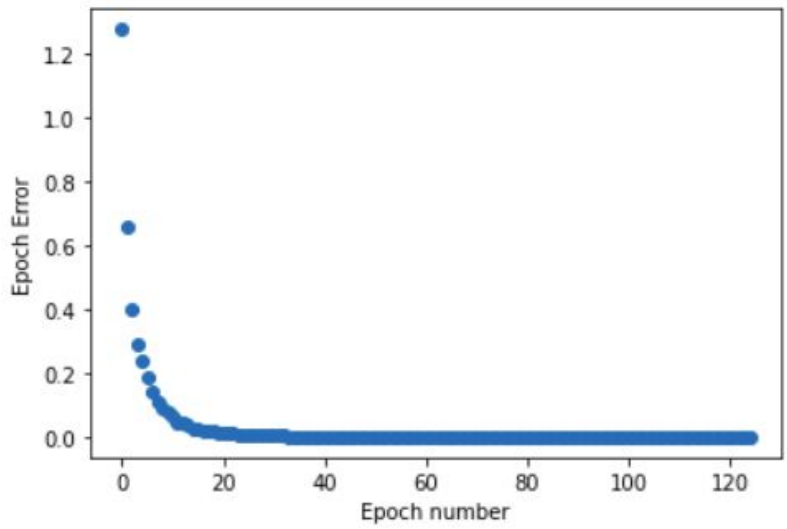
Parameter Learning: Parameters (w_{ij} , w_{jk}) learning is done by minimizing the error using the Gradient descent method.
To ensure that the learning rate is large in beginning and smaller later, we have used the following values for the learning rate:

epoch_number<50	epoch_number<200	else
learning rate = 0.1	earning rate = 0.001	learning rate = 0.0001

Activation function: Logistic Sigmoidal Function for hidden, Linear for output layer

Stopping Criteria: Threshold on the average error, which taken as 0.000001.

Cross-Validation:
Taking number of nodes in hidden layer(J) = 1
Considering that MSE for J = 1 is very less, we take 1 neuron in a hidden layer.
For the MLFFNN training, the average epoch error with epoch number is plotted as follows:

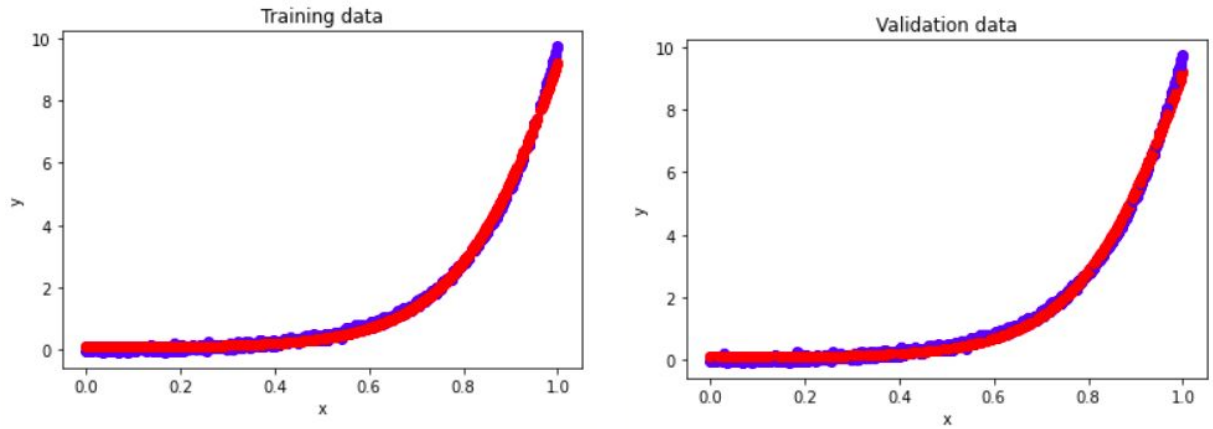


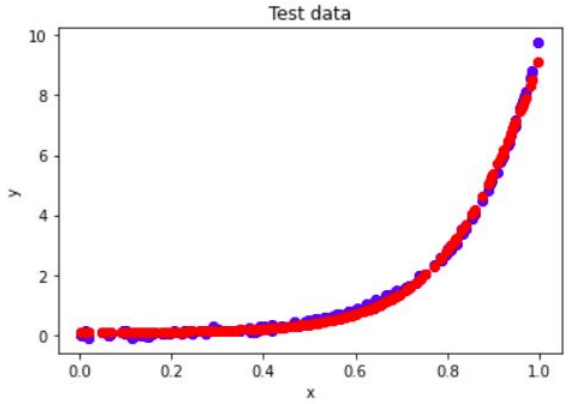
Mean Squared Error (MSE) on training data, validation data and test data:

Test Data	Validation Data	Train Data
0.0028644047732778112	0.002986945378419426	0.003160761407512309

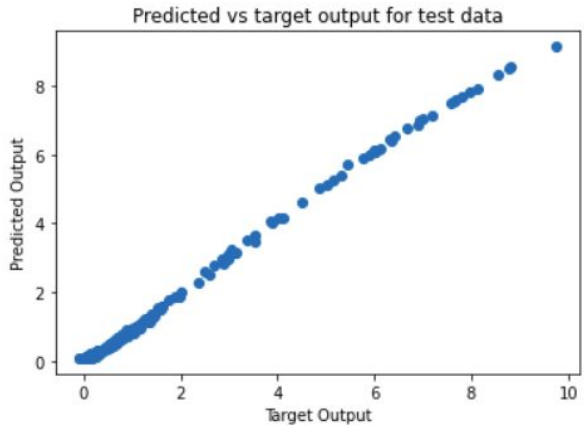
Plots of prediction made by model for training, test and validation data:

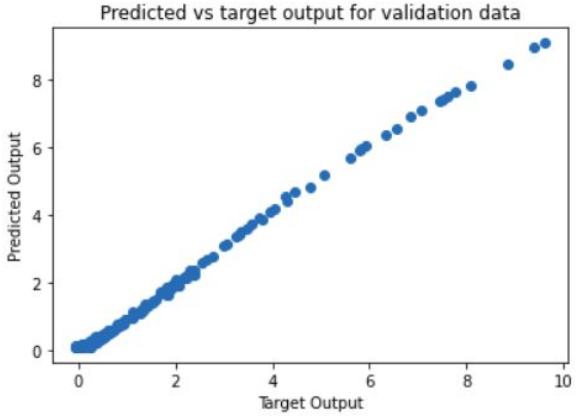
- Red colour indicates model prediction
- Blue colour indicates target output



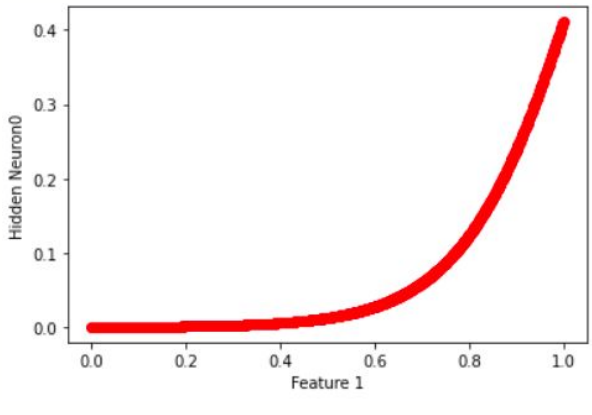


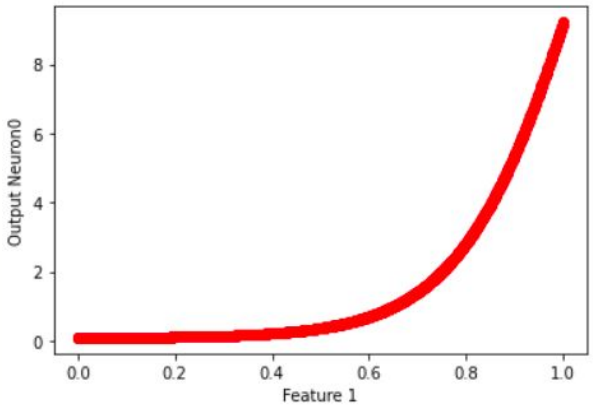
Plot of Predicted Output with Target Output:



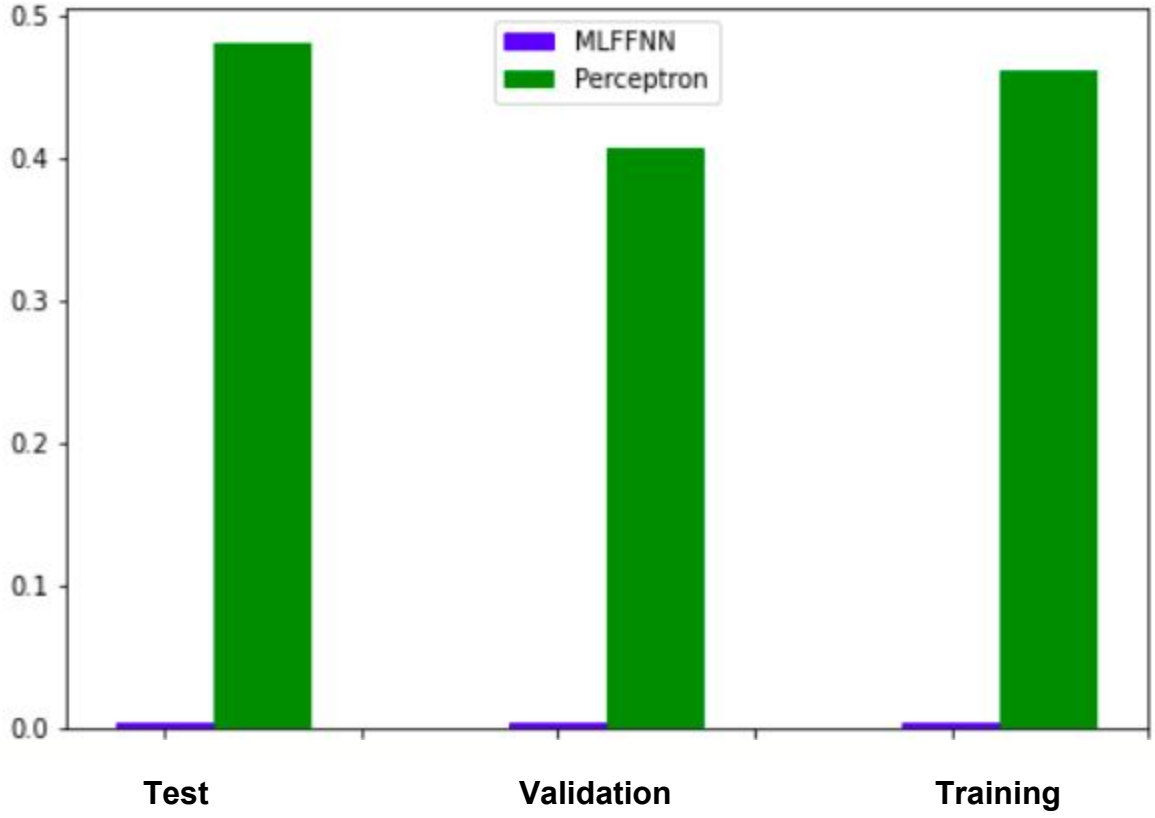


Output of each neuron after training:



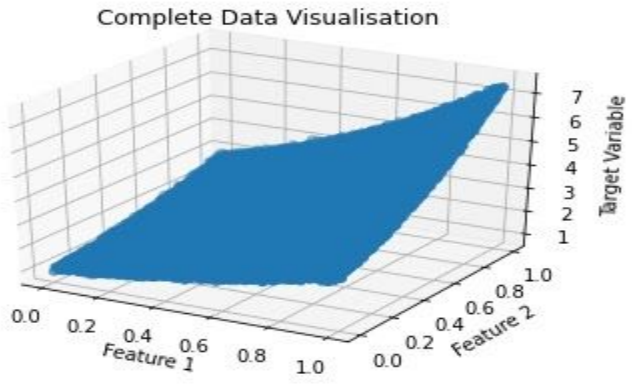


Model Comparison(MSE):



Certainly MLFFNN performs much better than perceptron.

Dataset 2: 2-dimensional (bivariate) input and 1-dimensional output

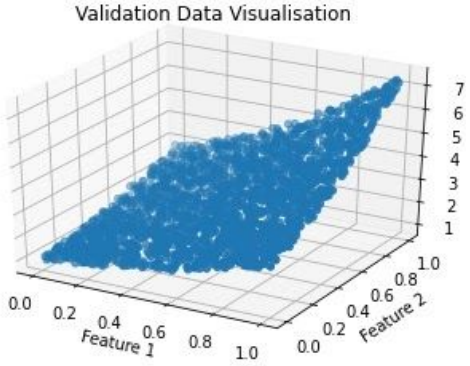


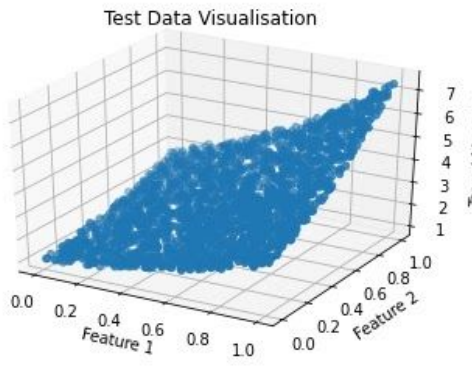
Pre Training: Since the range of data is fairly low, we haven't normalised the data.

Training Model: Perceptron

First, we divided data randomly into training, test and validation data with following proportion:

- Training: 60%
- Validation: 20%
- Test: 20%





Training:
Mode of Training: Pattern Mode

Weight initialization: Weights are generated randomly from the standard normal distribution.

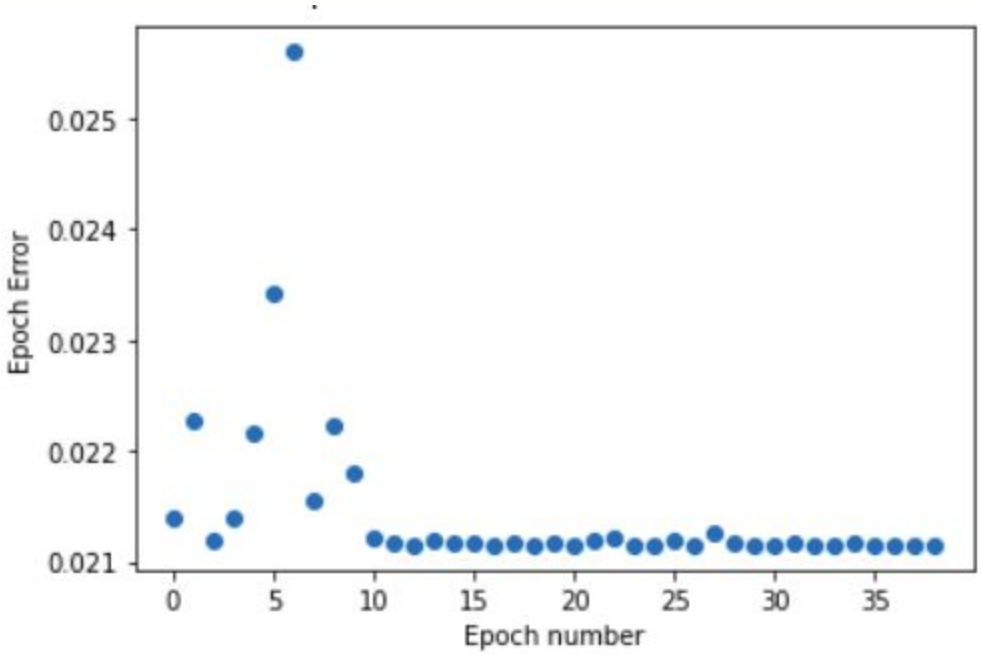
Parameter Learning: Parameter (w) learning is done by minimizing the error using the Gradient descent method.

To ensure that the learning rate is large in beginning and smaller later, we have used the following values for the learning rate:

epoch_number<50	epoch_number<200	else
learning rate = 0.1	earning rate = 0.001	learning rate = 0.0001

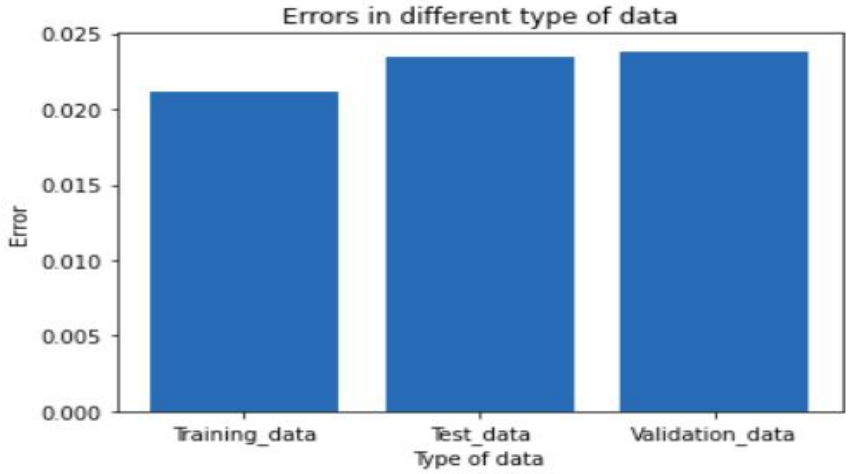
Activation function: Linear Function
Stopping Criteria: Threshold on the average error, which taken as 0.000001.
For 3 perceptrons training, the average epoch error with epoch number is plotted as follows:

Average Epoch Error with Epoch Number for perceptron training



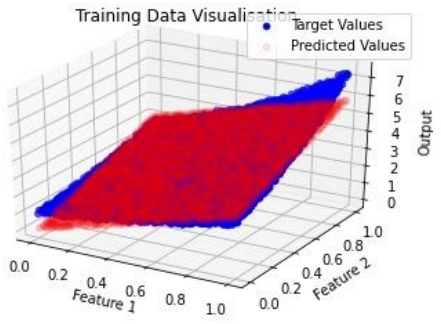
Mean Squared Error (MSE) on training data, validation data and test data:

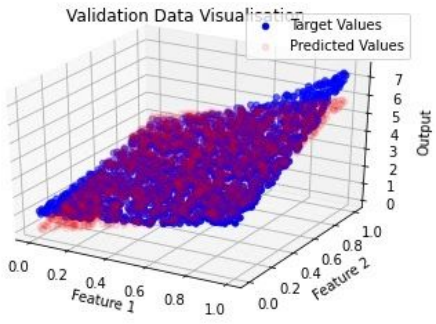
Training data	Validation data	Test data
0.02115	0.02352	0.02384

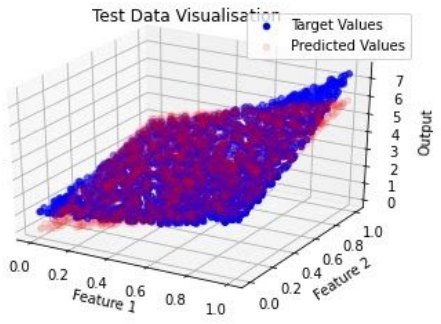


Plots of prediction made by model for training, test and validation data:

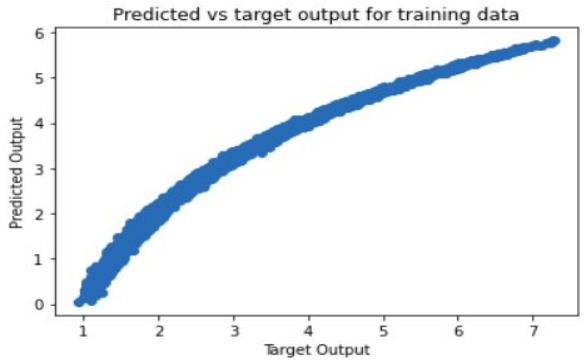
- Red colour indicates model prediction
- Blue colour indicates target output

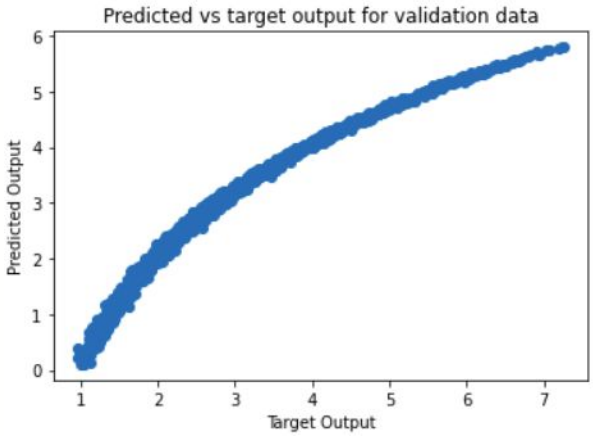


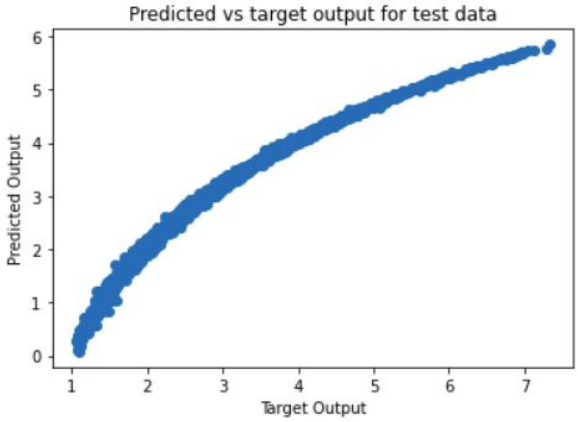




The plot of Predicted Output with Target Output:



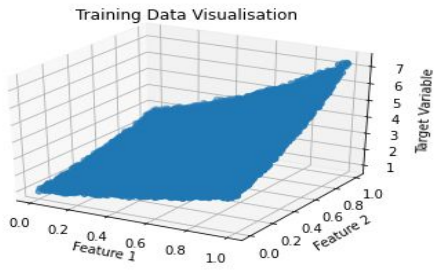


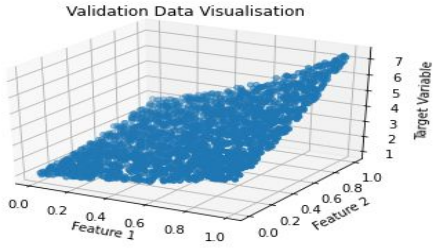


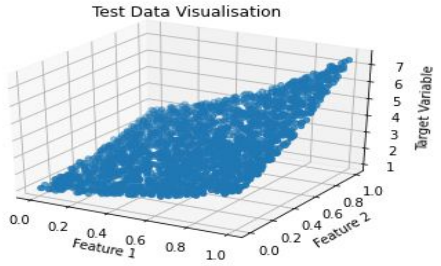
Training Model: MLFFNN (1 hidden layer)

First, we divided data randomly into training, test and validation data with following proportion:

- Training: 60%
- Validation: 20%
- Test: 20%







Training:

Mode of Training: Pattern Mode

Weight initialization: Weights are generated randomly from the standard normal distribution.

Parameter Learning: Parameters (w_{ij} , w_{jk}) learning is done by minimizing the error using the Gradient descent method.

To ensure that the learning rate is large in beginning and smaller later, we have used the following values for the learning rate:

epoch_number<50	epoch_number<200	else
learning rate = 0.1	earning rate = 0.001	learning rate = 0.0001

Activation function: Logistic Sigmoidal Function for hidden, Linear for output layer

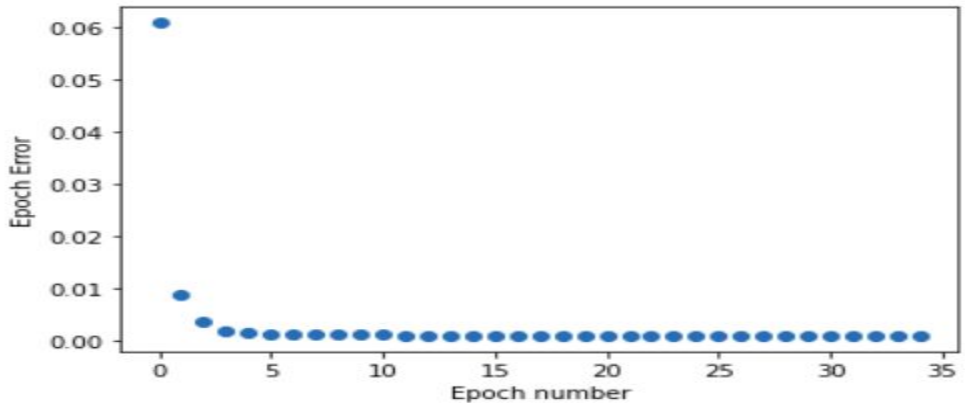
Stopping Criteria: Threshold on the average error, which taken as 0.000001.

Cross-Validation:

Taking J = 3

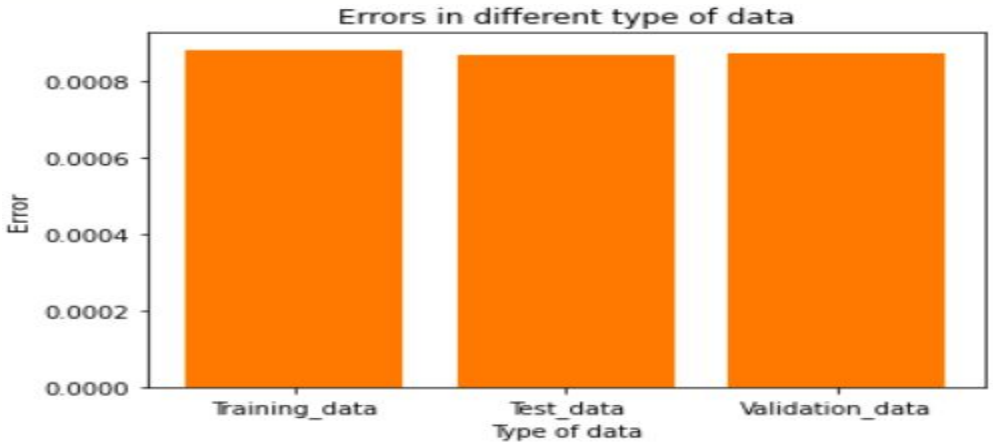
Considering that MSE for J = 3 is very less, we take 1 neuron in a hidden layer.

For the MLFFNN training, the average epoch error with epoch number is plotted as follows:



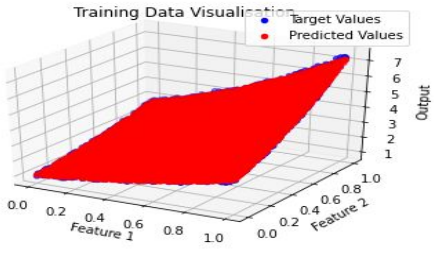
Mean Squared Error (MSE) on training data, validation data and test data:

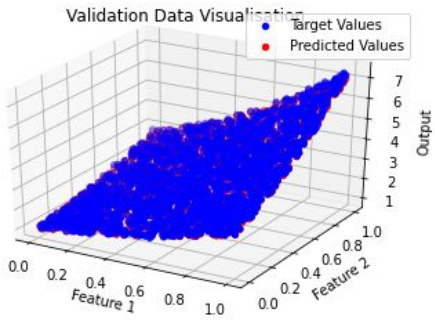
Test Data	**Validation Data**	**Train Data**
0.0008710709816624913	0.0008825771412914921	0.0008725146102872169

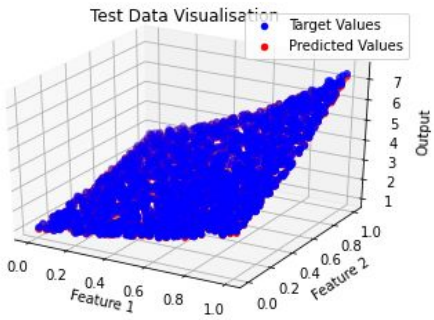


Plots of prediction made by model for training, test and validation data:

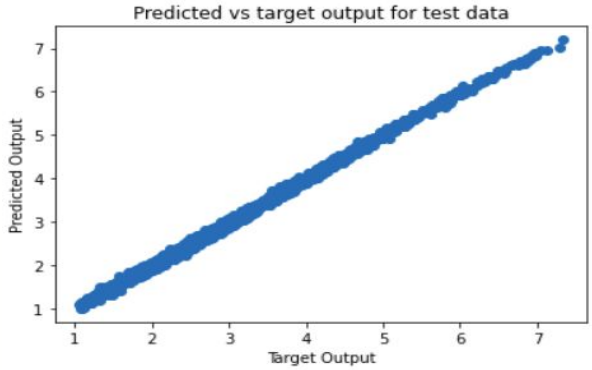
- Red colour indicates model prediction
- Blue colour indicates target output

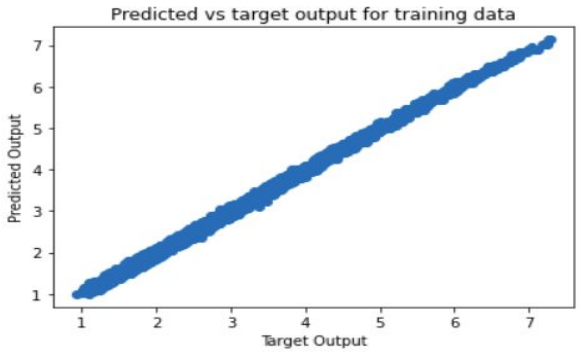


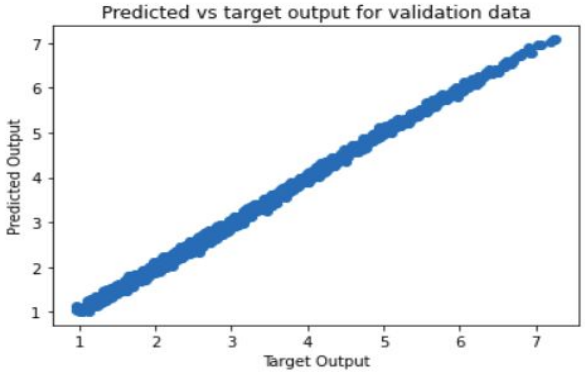




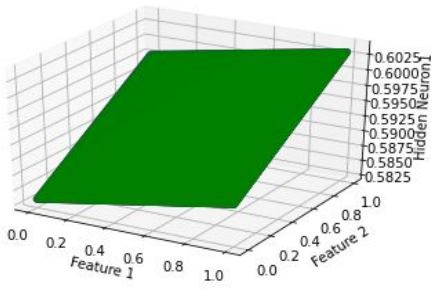
The plot of Predicted Output with Target Output:

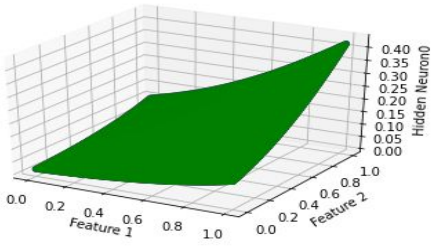


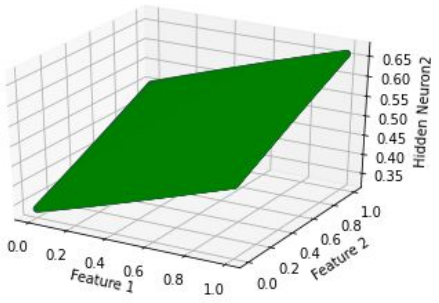


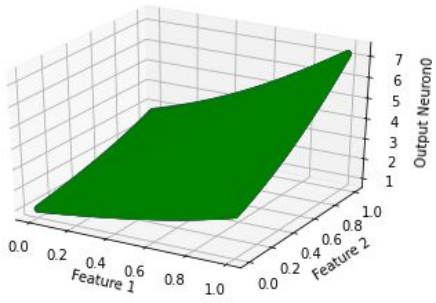


The output of each neuron after training:

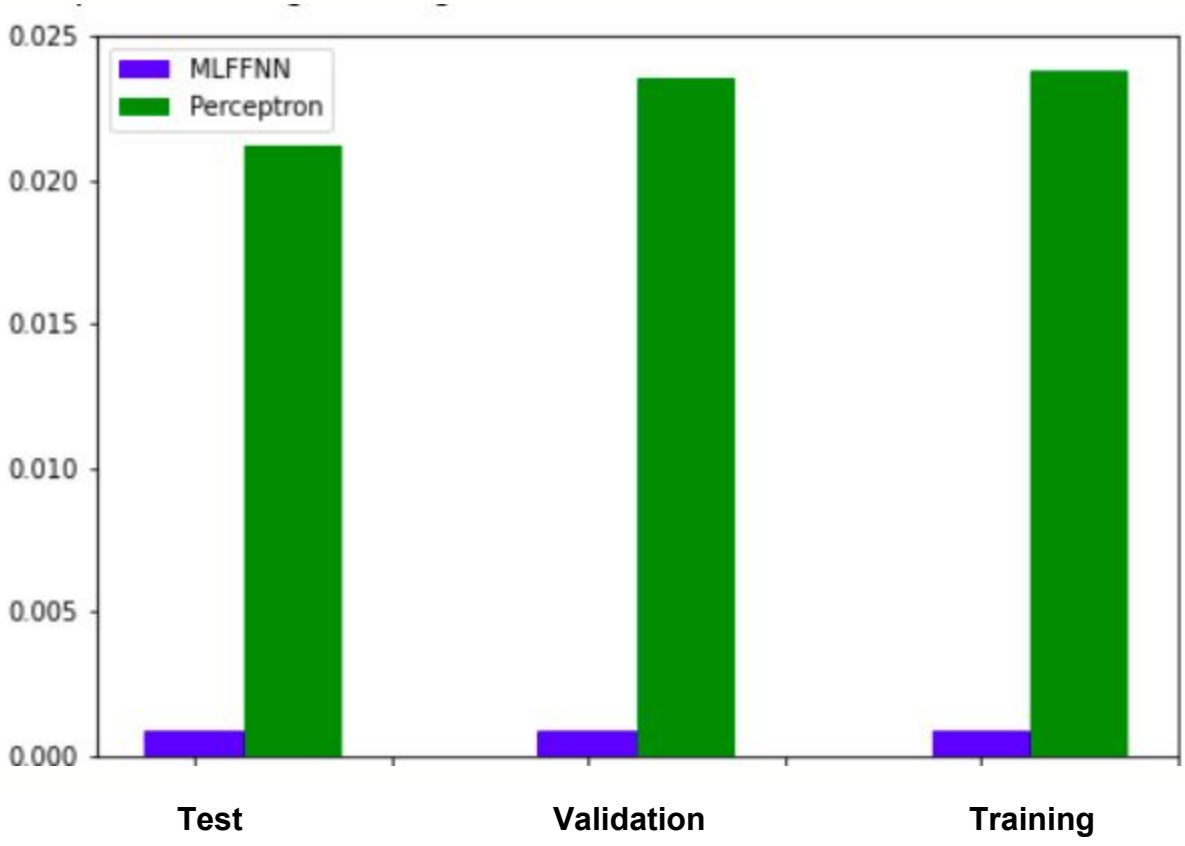








Model Comparison (MSE):



Certainly MLFFNN performs much better than perceptron considering the MSE of both models.